LAPORAN PRAKTIKUM DATA STRUCTURE

LINKED LIST

Dosen Pengampu:

H. Fatchurrochman,M.Kom

Asisten Praktikum:

Fillah Anjany 230605110033

Oleh :

Muhammad Alif Mujaddid **240605110082**

Jurusan Teknik Informatika Fakultas Sains dan Teknologi

UIN Maulana Malik Ibrahim Malang

2025

A. Pendahuluan

Linked list adalah struktur data linear, di mana elemen-elemennya tidak disimpan pada lokasi memori yang berurutan. Setiap elemen dalam linked list dihubungkan menggunakan pointer. Sederhananya, linked list terdiri dari node-node, di mana setiap node berisi data dan referensi (link) yang menunjuk ke node berikutnya dalam daftar.

Komponen Utama:

Node / Link, unit dasar dalam linked list, berisi:

- **Data**, nilai yang disimpan (misalnya angka atau teks).
- **Pointer (Next)**, alamat node berikutnya.

Head, pointer yang menunjukkan node pertama.

Tail (opsional), pointer yang menunjukkan node terakhir (dalam beberapa implementasi).

Ciri-Ciri Linked List:

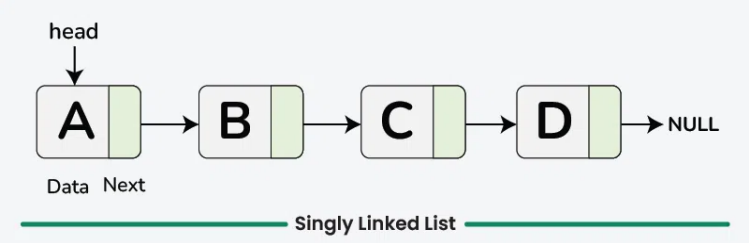
Dinamis (bisa bertambah/berkurang tanpa alokasi ulang seperti array).

Insert dan delete data lebih mudah daripada array, karena cukup mengubah pointer.

Akses data lebih lambat dibanding array, karena harus ditelusuri node satu per satu.

Jenis Jenis Linked List :

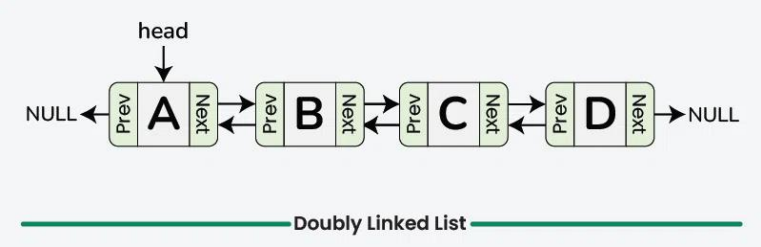
1. Singly Linked List



Tiap node hanya punya 1 pointer → menunjuk ke node berikutnya.

Traversal (penelusuran) cuma bisa dari depan ke belakang.

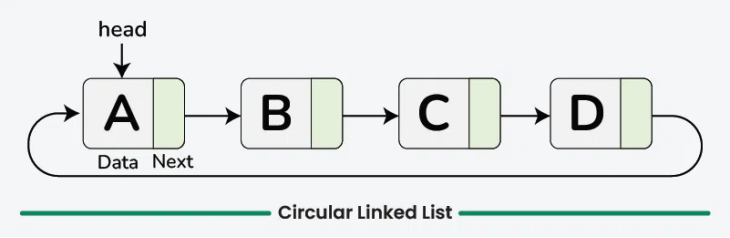
2. Doubly Linked List



Tiap node punya 2 pointer → satu menunjuk ke next, satu menunjuk ke previous.

Bisa ditelusuri bolak-balik.

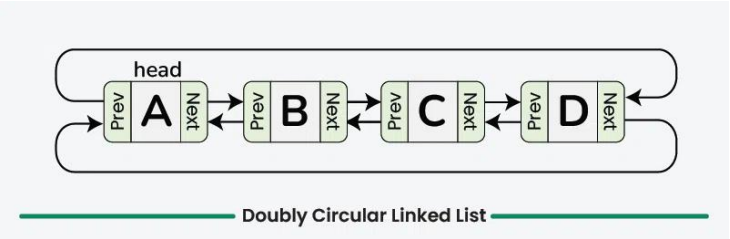
3. Circular Singly Linked List



Sama seperti singly, tapi node terakhir menunjuk kembali ke head.

Traversal bisa melingkar tanpa ujung.

4. Circular Doubly Linked List



Gabungan circular + doubly.
Node terakhir menunjuk kembali ke head, dan head menunjuk ke node terakhir.
Bisa ditelusuri dua arah dan berputar.

2. Berikut ini listing program yang menunjukkan implementasi linier singly-linked list

```
class Link {


    public int Data;
    public Link next;


    public Link(int Data) {
        this.Data = Data;
    }


    public void displayLink() {
        System.out.print(Data +" ");
    }
}


class LinkedList {


    private Link first;
    public LinkedList() {
        first = null;
    }


    public boolean isEmpty() {
        return (first == null);
    }


    public void insertFirst(int Data) {
        Link newLink = new Link(Data);
        newLink.next = first;
        first = newLink;
    }
```

```
public Link deleteFirst() {
    Link temp = first;
    first = first.next;
    return temp;
}


public Link find(int key) {
    Link current = first;
    while (current.Data != key) {
        if (current.next == null) {
            return null;
        } else {
            current = current.next;
        }
    }
    return current;
}


public Link delete(int key) {
    Link current = first;
    Link previous = first;
    while (current.Data != key) {
        if (current.next == null) {
            return null;
        } else {
            previous = current;
            current = current.next;
        }
  }
    if (current == first) {
        first = first.next;
    } else {
```

```
            previous.next = current.next;
        }
        return current;
    }

    public void displayList() {
        System.out.println("List(first
                            -->last):");
        Link current = first;
        while (current != null) {
            current.displayLink();
            current = current.next;
        }
        System.out.println("");
    }
}

public class LinkedListApp {
    public static void main(String[] args) {
        LinkedList theList = new LinkedList();

        theList.insertFirst(22);
        theList.insertFirst(44);
        theList.insertFirst(66);
        theList.insertFirst(88);
        theList.displayList();

        while (!theList.isEmpty()) {
            Link aLink =
                   theList.deleteFirst();

            System.out.print("Deleted ");
            aLink.displayLink();
```

```
            System.out.println("");
        }
        theList.displayList();

        theList.insertFirst(33);
        theList.insertFirst(55);
        theList.insertFirst(77);
        theList.insertFirst(88);
        theList.displayList();

        Link f = theList.find(77);
        if (f != null) {
            System.out.println("ketemu..."
                               + f.Data);
        } else {
            System.out.println("link tidak
                                ditemukan");
        }

        Link d = theList.delete(88);
        if (d != null) {
            System.out.println("hapus link
                    dengan key " + d.Data);
        } else {
            System.out.println("link tidak
                                ditemukan");
        }
        theList.displayList();
    }
}
```

Tuliskan output dari listing program tersebut!

```
addid@LAPTOP-F80MDN28 MINGW64 /a/Code/DSA (main)
$  /usr/bin/env A:\\Java\\jdk-22\\bin\\java.exe
ilsInExceptionMessages -cp C:\\Users\\addid\\App
_ws\\DSA_2f521fc6\\bin Praktikum.LinkedListApp
List(first-->last):
88
66
44
22

Deleted 88

Deleted 66

Deleted 44

Deleted 22

List(first-->last):

List(first-->last):
88
77
55
33

ketemu...77
hapus link dengan key 88
List(first-->last):
77
55
33
```

1. Class Link

Class Link berfungsi sebagai node dalam struktur data *linked list*. Setiap node menyimpan dua hal penting, yaitu data (int data) dan referensi ke node berikutnya (Link next). Konstruktor digunakan untuk menginisialisasi nilai data saat node dibuat, sementara next secara default bernilai null sampai dihubungkan dengan node lain. Method displayLink() berfungsi untuk menampilkan isi data dari node tersebut. Dengan demikian, class ini merupakan blok dasar yang membentuk rantai (list) dengan cara menghubungkan satu node ke node berikutnya.

2. Class LinkedList

Class LinkedList berfungsi sebagai wadah utama untuk mengelola node-node (Link). Ia menyimpan referensi ke node pertama (first) dan menyediakan berbagai operasi dasar untuk memanipulasi data dalam list. Method isEmpty() digunakan untuk mengecek apakah list kosong. insertFirst(int data) menambahkan node baru di depan list, sedangkan insertLast(int data) menambahkan node di akhir list. Operasi penghapusan juga disediakan, yaitu deleteFirst() untuk menghapus node paling depan, deleteLast() untuk menghapus node paling belakang, dan delete(int key) untuk menghapus node berdasarkan nilai tertentu. Selain itu, ada method find(int key) yang mencari node dengan nilai tertentu, serta displayList() yang menampilkan isi list dari

depan ke belakang. Class ini mengatur logika bagaimana node saling terhubung dan dimanipulasi.

3. Class LinkedListApp

Class LinkedListApp merupakan kelas driver atau program utama yang berfungsi untuk menguji semua operasi yang ada pada class LinkedList. Di dalam main(), pertama-tama list dibuat dan diisi menggunakan insertFirst(). Kemudian ditampilkan isinya dengan displayList(), setelah itu dilakukan penghapusan semua data dari depan dengan deleteFirst(). Program juga menambahkan data baru, melakukan pencarian node tertentu dengan find(), serta menghapus node dengan delete(key). Melalui class ini, kita bisa melihat cara kerja dan hasil nyata dari semua method yang sudah dibuat dalam LinkedList.

3. Pada class Link (listing nomor 2) terdapat deklarasi variable “next” yang bertipe objek Link (sama dengan nama class tersebut). Variable tersebut tidak di-inisialisasi dengan value tertentu. Jelaskan maksud dan fungsi dari variable next pada baris public Link next

Jawab :

public Link next adalah variabel referensi yang menunjuk ke node berikutnya dalam linked list. Kalau next = null, berarti node itu adalah node terakhir.

4. Jelaskan fungsi dan logika tiap method insertFirst(), deleteFirst(), find(), dan delete() yang terdapat pada listing nomer 2!

a. insertFirst(int data)

Fungsi:

Menambahkan node baru di **awal** linked list.

Logika:

1. Buat node baru (newLink) dengan data yang diberikan.
2. Atur newLink.next = first node baru menunjuk ke node pertama lama.
3. Geser pointer first ke newLink.

Hasilnya: node baru selalu berada di depan list.

b. deleteFirst()

Fungsi:

Menghapus node pertama dari linked list.

Logika:

1. Simpan node pertama ke temp.
2. Geser pointer first ke node berikutnya (first.next).
3. Node pertama lama otomatis "terlepas" dari list.
4. Return node yang dihapus (supaya bisa dipakai lagi kalau mau).

Hasilnya: node paling depan hilang, node kedua jadi node pertama.

c. find(int key)

Fungsi:

Mencari node berdasarkan nilai data (key).

Logika:

1. Mulai dari first, cek apakah current.data == key.
2. Kalau belum ketemu, pindah ke current.next.
3. Kalau sampai akhir (next == null) tapi tidak ketemu return null.
4. Kalau ketemu → return node tersebut.

Hasilnya: kembalikan node dengan data tertentu, atau null kalau tidak ada.

d. delete(int key)

Fungsi:

Menghapus node berdasarkan nilai data (key).

Logika:

1. Cari node yang datanya sesuai dengan key.

Simpan juga previous (node sebelum current).

2. Jika current adalah node pertama (first) geser first ke node berikutnya.
3. Jika current di tengah/akhir sambungkan previous.next ke current.next (melewati node yang dihapus).
4. Return node yang dihapus.

Hasilnya: node dengan nilai tertentu dihapus dari list.

5. Jelaskan langkah/logika untuk menambahkan operasi “insert Last” yang berfungsi untuk menambahkan data pada akhir list. Tuliskan listing program yang perlu ditambahkan pada program nomor 2!

   Logika insertLast()

   1. Buat node baru dengan data yang diberikan.
   2. Jika list **kosong** (first == null), maka node baru langsung jadi node pertama (first).
   3. Jika list **tidak kosong**, lakukan traversal (penelusuran) dari first sampai menemukan node terakhir (current.next == null).
   4. Set current.next = newLink, sehingga node terakhir lama sekarang menunjuk ke node baru.

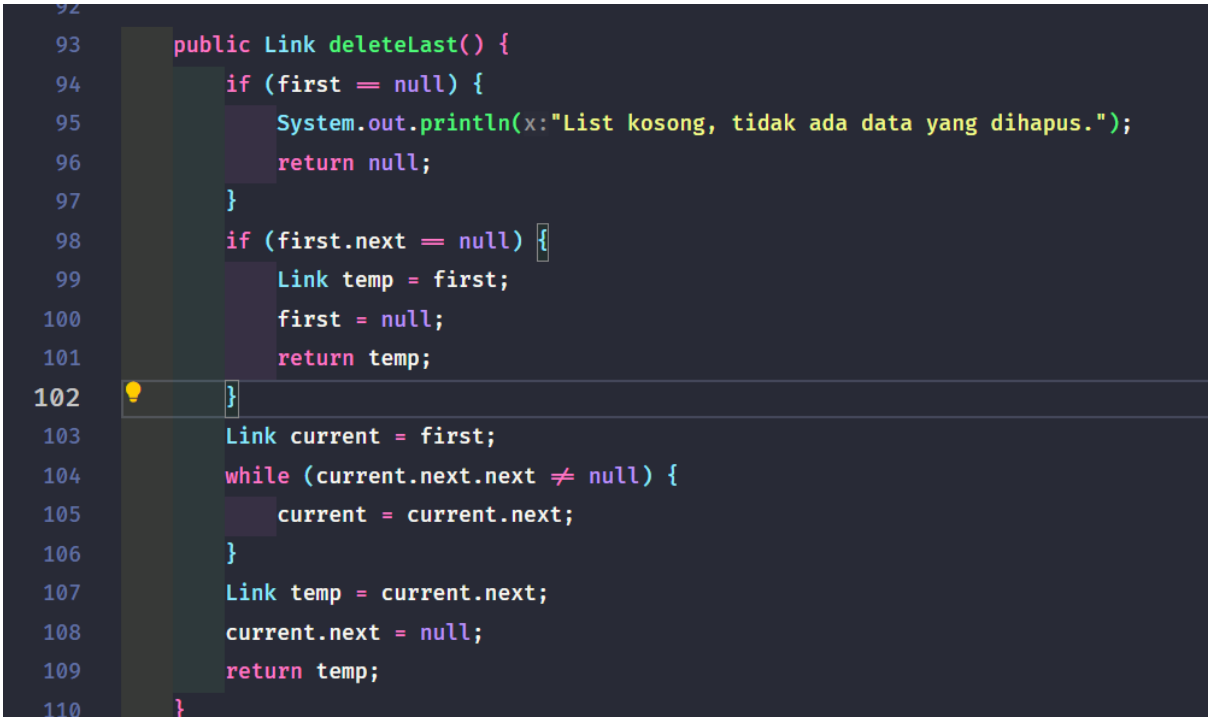
```
79
80      public void insertLast(int data) {
81          Link newLink = new Link(data);
82          if (first == null) {
83              first = newLink;
84          } else {
85              Link current = first;
86              while (current.next ≠ null) {
87                  current = current.next;
88              }
89              current.next = newLink;
90          }
91      }
```

6. Jelaskan langkah/logika untuk menambahkan operasi “delete Last” yang berfungsi untuk menghapus data pada akhir list. Tuliskan listing program yang perlu ditambahkan pada program nomor 2!

Langkah/Logika Delete Last pada Singly Linked List

Operasi deleteLast() artinya kita ingin **menghapus node terakhir** dari list. Karena singly linked list **hanya punya pointer next (tidak punya pointer ke previous)**, maka untuk menghapus node terakhir:

1. Cek apakah list kosong → jika kosong, tidak ada yang dihapus.
2. Jika hanya ada **1 node** → hapus node tersebut, lalu first = null.
3. Jika ada lebih dari 1 node:
   - Gunakan pointer current untuk menelusuri list.
   - Berhenti ketika menemukan node **sebelum node terakhir** (current.next.next == null).
   - Set current.next = null, sehingga node terakhir terhapus.



```java
public Link deleteLast() {
    if (first == null) {
        System.out.println(x:"List kosong, tidak ada data yang dihapus.");
        return null;
    }
    if (first.next == null) {
        Link temp = first;
        first = null;
        return temp;
    }
    Link current = first;
    while (current.next.next != null) {
        current = current.next;
    }
    Link temp = current.next;
    current.next = null;
    return temp;
}
```

7. Berikut ini listing program yang menunjukkan implementasi doubly-linked list

```
class Link {
    public int Data;
    public Link next;
    public Link previous;

    public Link(int Data) {
        this.Data = Data;
    }
    public void displayLink() {
        System.out.print(Data + " ");
    }
}
```

```
class DoublyLinkedList {
    private Link first;
    private Link last;

    public DoublyLinkedList() {
        first = null;
        last = null;
    }
    public boolean isEmpty() {
        return first == null;
    }
    public void insertFirst(int Data) {
        Link newLink = new Link(Data);
        if (isEmpty()) {
            last = newLink;
        } else {
            first.previous = newLink;
        }
        newLink.next = first;
        first = newLink;
    }
    public void insertLast(int Data) {
        Link newLink = new Link(Data);
        if (isEmpty()) {
            first = newLink;
        } else {
            last.next = newLink;
            newLink.previous = last;
        }
        last = newLink;
    }
```

```
public Link deleteFirst() {
    Link temp = first;
    if (first.next == null) {
        last = null;
    } else {
        first.next.previous = null;
    }
    first = first.next;
    return temp;
}
```

```
public Link deleteLast() {
    Link temp = last;
    if (first.next == null) {
        first = null;
    } else {
        last.previous.next = null;
    }
    last = last.previous;
    return temp;
}
```

```
public boolean insertAfter(int key,
                           int Data) {
    Link current = first;
    while (current.Data != key) {
        current = current.next;
        if (current == null) {
            return false;
        }
    }
    Link newLink = new Link(Data);
    if (current == last) {
```

```
            newLink.next = null;
            last = newLink;
        } else {
            newLink.next = current.next;
            current.next.previous = newLink;
        }
        newLink.previous = current;
        current.next = newLink;
        return true;
    }
    public Link deleteKey(int key) {
        Link current = first;
        while (current.Data != key) {
            current = current.next;
            if (current == null) {
                return null;
            }
        }
        if (current == first) {
            first = current.next;
        } else {
            current.previous.next =
                                current.next;
        }
        if (current == last) {
            last = current.previous;
        } else {
            current.next.previous =
                            current.previous;
        }
        return current;
    }
    public void displayForward() {
        System.out.print("List "
```

```
                        + "(first-->last): ");
        Link current = first;
        while (current != null) {
            current.displayLink();
            current = current.next;
        }
        System.out.println("");
    }
    public void displayBackward() {
        System.out.print("List "
                        + "(last-->first): ");
        Link current = last;
        while (current != null) {
            current.displayLink();
            current = current.previous;
        }
        System.out.println("");
    }
} // akhir class
```

```
public class DoublyLinkedListApp {
    public static void main(String[] args) {
        DoublyLinkedList theList =
                    new DoublyLinkedList();
        theList.insertFirst(22);
        theList.insertFirst(44);
        theList.insertFirst(66);
        theList.displayForward();
        theList.insertLast(11);
        theList.insertLast(33);
        theList.insertLast(55);
        theList.displayForward();
        theList.displayBackward();
        theList.deleteFirst();
```

Tuliskan output program ini:

```
        theList.displayForward();
        theList.deleteLast();
        theList.displayForward();
        theList.deleteKey(11);
        theList.displayForward();
        theList.insertAfter(22, 77);
        theList.insertAfter(33, 88);
        theList.displayForward();
    }
}
```

Output :

```
list first --> last
66 44 22
list first --> last
66 44 22 11 33 55
List (last-->first):
55 33 11 22 44 66
list first --> last
44 22 11 33 55
list first --> last
44 22 11 33
list first --> last
44 22 33
list first --> last
44 22 77 33 88
```

Class Link

Class Link berfungsi sebagai node dalam *doubly linked list*. Setiap node menyimpan sebuah data (Data) dan dua referensi yaitu next untuk menunjuk ke node berikutnya serta previous untuk menunjuk ke node sebelumnya. Dengan adanya dua pointer ini, list bisa ditelusuri maju maupun mundur. Method displayLink() digunakan untuk menampilkan isi data dari node.

Class DoublyLinkedList

Class DoublyLinkedList merupakan inti dari struktur data. Class ini menyimpan referensi ke node pertama (first) dan node terakhir (last), serta menyediakan operasi untuk memanipulasi isi list. Method insertFirst() digunakan untuk menambahkan node baru di depan list, sedangkan insertLast() menambahkan node di bagian akhir. Untuk operasi penghapusan, tersedia deleteFirst() untuk menghapus node pertama dan deleteLast() untuk menghapus node terakhir. Selain itu, terdapat method insertAfter() yang dapat menyisipkan node baru setelah node tertentu, serta deleteKey() yang menghapus node dengan nilai tertentu. Method displayForward() menampilkan isi list dari depan ke belakang, dan displayBackward() menampilkan isi list dari belakang ke depan.

Class DoublyLinkedListApp

Class DoublyLinkedListApp adalah program utama yang digunakan untuk menguji semua operasi pada DoublyLinkedList. Di dalam method main(), program melakukan serangkaian percobaan mulai dari menambahkan node di depan dan belakang list, menampilkan isi list ke dua arah, menghapus node pertama dan terakhir, menghapus node berdasarkan nilai tertentu, hingga menyisipkan node setelah node tertentu. Setiap operasi ditampilkan hasilnya ke layar sehingga perubahan isi list dapat diamati secara bertahap.

8. Dari penjelasan tiap bagian yang Anda tuliskan pada nomor 7, tuliskan kesimpulan logika yang digunakan pada tiap method: insertFirst(), insertLast(), insertAfter(), deleteFirst(), deleteLast(), deleteKey()!

1. insertFirst(int Data)

- Buat node baru (newLink).
- Jika list kosong node baru jadi first sekaligus last.
- Jika tidak kosong hubungkan node baru ke first, lalu first.previous diarahkan ke node baru.
- Update first = newLink.

  Logika: menambahkan node di depan list.

2. insertLast(int Data)

- Buat node baru (newLink).
- Jika list kosong node baru jadi first.
- Jika tidak kosong hubungkan last.next ke node baru, lalu newLink.previous = last.
- Update last = newLink.

  Logika: menambahkan node di belakang list.

3. insertAfter(int key, int Data)

- Cari node dengan Data == key mulai dari first.
- Jika tidak ketemu return false.
- Jika ketemu:
  - Buat node baru (newLink).
  - Jika current == last node baru jadi last.
  - Jika bukan last hubungkan newLink di antara current dan current.next.
- Return true.

  Logika: menyisipkan node setelah node dengan key tertentu.

4. deleteFirst()

- Simpan node first ke temp.
- Jika hanya ada 1 node last = null.
- Jika lebih dari 1 set first.next.previous = null.
- Geser first = first.next.

  Logika: menghapus node paling depan.

5. deleteLast()

- Simpan node last ke temp.
- Jika hanya ada 1 node first = null.
- Jika lebih dari 1 set last.previous.next = null.

- Geser last = last.previous.
  Logika: menghapus node paling belakang.

6. deleteKey(int key)

- Cari node dengan Data == key.
- Jika tidak ketemu return null.
- Jika ketemu:
  - Jika node adalah first update first = current.next.
  - Jika node adalah last update last = current.previous.
  - Jika di tengah hubungkan previous dan next dari node yang akan dihapus.
- Return node yang dihapus.
  Logika: menghapus node dengan nilai tertentu, bisa di depan, tengah, atau belakang.